PCT

WORLD INTELLECTUAL PROPER
International Bur

INTERNATIONAL APPLICATION PUBLISHED UNDER

WO 9605556A1

(51) International Patent Classification 6 : G06F 11/00

A1

(11) International Publication Number: WO 96/05556

(43) International Publication Date: 22 February 1996 (22.02.96)

(21) International Application Number: PCT/US95/09691

(22) International Filing Date: 9 August 1995 (09.08.95)

(30) Priority Data:
289,148 10 August 1994 (10.08.94) US

(71) Applicant: INTRINSA CORPORATION [US/US]; 101 University Avenue, Palo Alto, CA 94301 (US).

(72) Inventors: HALEY, Matthew, A.; 3929 Woodcreek Lane, San Jose, CA 95117-3445 (US). PINCUS, Jonathan, D.; 4026 18th Street, San Francisco, CA 94114 (US). BUSH, William, R.; 1739 Lexington Avenue, San Mateo, CA 94402 (US).

(74) Agent: MACPHERSON, Alan, H.; Skjerven, Morrill, MacPherson, Franklin & Friel, 25 Metro Drive, Suite 700, San Jose, CA 95110 (US).

(81) Designated States: AM, AT, AU, BB, BG, BR, BY, CA, CH, CN, CZ, DE, DK, EE, ES, FI, GB, GE, HU, IS, JP, KE, KG, KP, KR, KZ, LK, LR, LT, LU, LV, MD, MG, MN, MW, MX, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, TJ, TM, TT, UA, UG, UZ, VN, European patent (AT, BE, CH, DE, DK, ES, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, ML, MR, NE, SN, TD, TG), ARIPO patent (KE, MW, SD, SZ, UG).

Published

With international search report.
Before the expiration of the time limit for amending the claims and to be republished in the event of the receipt of amendments.

(54) Title: COMPUTER PROCESS RESOURCE MODELLING METHOD AND APPARATUS

(57) Abstract

An error detection mechanism for detecting programming errors in a computer program. A component of the computer program, e.g., a procedure or function of the computer program, is analyzed to determine the effect of the component on resources used by the computer program. A component is analyzed by traversing the computer instructions, i.e., statements, of the component and tracking the state of resources used by the components as affected by the statements of the component. Each resource has a prescribed behavior represented by a number of states and transition between states. Violations in the prescribed behavior of a resource resulting from an emulated execution of the statements of the component are detected and reported as programming errors. Resources used by two or more components are modelled by modelling externals of the components. The effect of execution of a component on externals and resources of the component is determined by traversing one or more possible control flow paths through the component and tracking the use of each external and resource by each statement of each control flow path. Once the effect of execution of a component on externals and resources of the component is determined, a model of the component is created and used to model externals and resources of other components which invoke the modelled component.

FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AT	Austria	GB	United Kingdom	MR	Mauritania
AU	Australia	GE	Georgia	MW	Malawi
BB	Barbados	GN	Guinea	NE	Niger
BE	Belgium	GR	Greece	NL	Netherlands
BF	Burkina Faso	HU	Hungary	NO	Norway
BG	Bulgaria	IE	Ireland	NZ	New Zealand
BJ	Benin	IT	Italy	PL	Poland
BR	Brazil	JP	Japan	PT	Portugal
BY	Belarus	KE	Kenya	RO	Romania
CA	Canada	KG	Kyrgystan	RU	Russian Federation
CF	Central African Republic	KP	Democratic People's Republic of Korea	SD	Sudan
CG	Congo			SE	Sweden
CH	Switzerland	KR	Republic of Korea	SI	Slovenia
CI	Côte d'Ivoire	KZ	Kazakhstan	SK	Slovakia
CM	Cameroon	LI	Liechtenstein	SN	Senegal
CN	China	LK	Sri Lanka	TD	Chad
CS	Czechoslovakia	LU	Luxembourg	TG	Togo
CZ	Czech Republic	LV	Latvia	TJ	Tajikistan
DE	Germany	MC	Monaco	TT	Trinidad and Tobago
DK	Denmark	MD	Republic of Moldova	UA	Ukraine
ES	Spain	MG	Madagascar	US	United States of America
FI	Finland	ML	Mali	UZ	Uzbekistan
FR	France	MN	Mongolia	VN	Viet Nam
GA	Gabon				

COMPUTER PROCESS RESOURCE MODELLING METHOD AND APPARATUS

REFERENCE TO APPENDIX A

Appendix A, which is a part of this disclosure, is a list of computer programs and related data in one embodiment of the present invention, which is described more completely below.

A portion of the disclosure of this patent document contains material which is subject to copyright protection. The copyright owner has no objection to the facsimile reproduction by anyone of the patent document or the patent disclosure, as it appears in the Patent and Trademark Office patent files or records, but otherwise reserves all copyright rights whatsoever.

BACKGROUND OF THE INVENTION

Field of the Invention

The present invention relates to the analysis of computer programs and, in particular, to the detection of programming errors in a computer program through analysis of the use of resources prescribed by the computer program.

Discussion of Related Art

Some existing programming error detection methods detect violations in the computer instruction protocol with which a particular program comports. Such a programming error detection method is called "static checking" since the syntax of the computer instructions, or "statements", of the computer program is analyzed outside the context of the behavior

resulting from the execution of those statements. The term "statement" is used herein as it is defined in Section 6.6 of American National Standard for Programming Languages--C (American National Standards Institute/International Organization for Standardization ANSI/ISO 9899-1990), which is reproduced in Herbert Schildt, The Annotated ANSI C Standard, (Osborne McGraw-Hill 1990) (hereinafter the C Standard). Briefly, in the context of the C computer language, a statement is a computer instruction other than a declaration. In other words, a statement is a any expression or instruction which directs a computer to carry out one or more processing steps. Static checking in the context of the C computer language includes, for example, (i) making sure that no two variables in the computer program are identified by the same name; (ii) ensuring that each "break" statement corresponds to a preceding "while", "for", or "switch" statement; and (iii) verifying that operators are applied to compatible operands. Static checking is discussed, for example, in Alfred V. Aho et al., Compilers, (Addison Wesley 1988).

Some existing static checking methods, which are generally called "data flow analysis" techniques, analyze data flow through a program to detect programming errors. Such analysis includes use of control flow information, such as sequencing of statements and loop statements, to detect the improper use of data objects, e.g., the use of a variable before a value has been assigned to the variable. Flow of control in a computer program is the particular sequence in which computer instructions of the computer program are executed in a computer process defined by the computer program. Computer programs and processes and the relation therebetween are discussed more completely below. Data flow techniques are discussed

in Beizer, Software Testing Techniques, (1990) at pp. 145-172.

Existing static checking techniques suffer from the inability to track use of resources through several discrete components of a computer program such as several functions which collectively form a computer program. For example, a variable may be initialized in a first function and used in a calculation in a second, subsequently executed function. By analysis of only the computer instructions of the second function, the variable appears to be used before the variable is initialized which can be erroneously reported as an error. In addition, existing static checking techniques are static in nature and do not consider particular data values associates with particular data objects. Static analysis is limited to what can be determined without considering the dynamic effects of program execution. Beizer describes several areas for which static analysis is inadequate, including: arrays, especially dynamically calculated indices and dynamically allocated arrays; records and pointers; files; and alternate state tables, representing the different semantics of different types in the same program.

Static checkers do not detect errors involving calculated addresses corresponding to dynamically allocated memory or calculated indices into arrays. Calculated addresses and indices are addresses and indices, respectively, which are calculated during the execution of a computer process. Static checkers do not detect such errors in a computer program because checking for such errors typically involves determining the precise values of calculated addresses and indices, which in turn involves consideration of the behavior of the computer program during execution, i.e., as a computer process.

Static checkers do not detect errors involving the use of questionably allocated resources or the use of resources whose state is determined by the value of a variable or other data object. In the C computer language, a resource, e.g., dynamically allocate memory or a file, is questionably allocated. In other words, a function which allocates the resource completes successfully, even if allocation of the resource failed. Whether the allocation succeeded is determined by comparison of the returned item of the function, which is a pointer to the allocated resource, to an invalid value, e.g., NULL. Static checkers do not consider the behavior of a called function but instead only verify that the syntax of the call to the called function comports with the syntax prescribed in the particular computer language. Therefore, static checkers do not detect errors involving use of a resource which is questionably allocated.

As described above, a static checker does not consider the behavior of a called function. Thus, verifying the use of a resource which spans multiple functions is impossible. For example, if a first function allocates a resource, a second function uses the resource, and a third function deallocates the resource, static checking of any of the first, second, and third functions alone or a function calling all three functions, cannot verify the proper use of the resource.

When using an error detection technique, which employs insufficient information regarding the behavior of a computer program during execution, the errors reported by such a technique are either under-inclusive or over-inclusive. For example, if a function accepts as a parameter a pointer to an allocated resource, e.g., a file, and uses the parameter without comparing the parameter to an invalid pointer, the function

contains a possible error. Whether the function contains an error depends on circumstances which are unknown within the context of the function. For example, if the pointer is verified to be a valid pointer before the function is called, there is no error in the function. To report the use of the pointer as an error would clutter an analysis of the function with a falsely reported error, and thus would be over-inclusive. Falsely reporting errors in analysis of a large program, at best, is an inconvenience to a program developer and, at worst, renders analysis of a computer program useless. If the pointer is not checked to be valid prior to calling the function, failure to report the error results in failure to detect an error which can cause an execution of the computer program to be aborted abruptly and can result in the corruption of data structures and possibly in the loss of valuable data.

One particular drawback of the failure of static checking techniques to consider the dynamic behavior of a computer program is the reporting of apparent, but "false", errors, i.e., errors resulting from computer instructions through which control cannot flow. In functions in which control flow paths depend on particular values associated with particular data structures and program variables, control flow cannot be determined without considering the values associated with those data structures and variables which generally in turn cannot be determined without consideration of the behavior of the function during execution. As a result, instructions which are not executed or which are executed only under specific circumstances are generally assumed to always be executed by static checkers.

Another type of existing programming error detection technique is called program verification. In

program verification, a computer program is treated as a formal mathematical object. Errors in the computer program are detecting by proving, or failing to prove, certain properties of the computer program using theoretical mathematics. One property for which a proof is generally attempted is that, given certain inputs, a computer process defined by the computer program produces certain outputs. If the proof fails, the computer program contains a programming error. Such program verification techniques are described, for example, in Eric C.R. Hehner et al., A Practical Theory of Programming, (Verlag 1993) and Ole-Johan Dahl, Verifiable Programming, (Prentice Hall 1992).

Verified programming techniques are limited in at least two ways: (i) only properties of computer programs which can be expressed and automatically proven using formal logic can be verified, and (ii) a person developing a computer program generally must formally specify the properties of the computer program. Formally specifying the properties of a computer program is extremely difficult in any case and intractable for larger programs. As a result, commercially successful products employing verified programming techniques are quite rare.

In another type of programming error detection technique, a computer program is executed, thus forming a computer process, and the behavior of the computer process is monitored. Since a computer program is analyzed during execution, such a programming error detection technique is called "runtime checking". Some runtime checking techniques include automatically inserting computer instructions into a computer program such that execution of the inserted computer instructions note, during execution of the computer program, the status of variables and resources of the

computer program. Such an error detection technique is described by U.S. Patent Number 5,193,180 to Hastings.

Runtime checking can typically detect errors such as array indices out of bounds and memory leaks. Examples of runtime checking include Purify which is available from Pure Software Inc. of Sunnyvale, California and Insight which is available from Parasoft Corporation of Pasadena, California. Purify inserts into a computer program monitoring computer instructions after a computer program has been compiled in to an object code form, and Insight inserts into a computer program monitoring computer instructions before a computer program is compiled, i.e., while the computer program is still in a source code form.

Runtime checking is generally limited to what can be determined by actually executing the computer instructions of a computer program with actual, specific inputs. Runtime checking does not consider all possible control flow paths through a computer program but considers only those control flow paths corresponding to the particular inputs to the computer program supplied during execution. It is generally impracticable to coerce a computer process, formed by execution of the computer instructions of a computer program, to follow all possible control flow paths. To do so requires that a programmer anticipate all possible contingencies which might occur during execution of the computer instructions of a computer program and to cause or emulate all possible combinations of occurrences of such contingencies.

Furthermore, runtime checking can only be used when the computer program is complete. Analysis of a single function before the function is incorporated into a complete program is impossible in runtime checking since the function must be executed to be analyzed. Analysis of a function using runtime

checking therefore requires that (i) all functions of a computer program be developed and combined to form the computer program prior to analysis of any of the functions or (ii) that a special purpose test program, which incorporates the function, be developed to test the function. Top-down programming, which involves the design, implementation, and testing of individual functions prior to inclusion in a complete computer program and which is a widely known and preferred method of developing more complex computer programs, therefore does not lend itself well to runtime analysis.

What is needed is a programming error detection technique which considers the dynamic behavior of a computer program, which automatically considers substantially all possible control flow paths through the computer program, and which does not require a programmer of such a computer program to express the computer program in an alternative, e.g., mathematical, form. What is further needed is a programming error detection technique which analyzes an individual component of a program, considering the behavior of the component during execution. What is further needed is a programming error detection technique which considers the behavior of a component whose execution is invoked by a computer program component under analysis.

SUMMARY OF THE INVENTION

In accordance with the present invention, a computer program is analyzed, and programming errors in the computer program are detected, by modelling the behavior of resources used by the computer program and detecting potential state violations in the those resources. A resource is modelled according to resource states and resource state transitions which describe the behavior of the resource. The computer

instructions of the computer program are dynamically inspected, i.e., the dynamic behavior of the computer instructions is determined and the states of resources are changed according to the dynamic behavior of the computer instructions.

Each component of a computer program is analyzed individually. Use of a resource whose use spans more than one component, e.g., a resource which is allocated by a first component, used by a second component and deallocated by a third component, is analyzed by modelling the externals of each component. Two components of a computer program communicate with one another through the externals of each component. For example, information regarding a resource allocated by a first component is transmitted to a second component, which uses the resource, through the externals of the first and second components. By analyzing the behavior of each component with respect to the externals of the component, resources whose use span more than one component are properly modelled.

Each component is analyzed and the effect of execution of the component on each external of the component is determined. From the analysis of the component, a model of the component is created. The model of the component describes the effect of execution of the component on each external of the component in terms of changes in the respective states of the externals and the introduction of new resources associated with any external of the component. Execution of the modelled component can have any of a number of effects on any individual external, and those effects are represented in a composite state of the external. The model of the component can then be used in the analysis of other components which invoke execution of the modelled component.

INTERNATIONAL SEARCH REPORT

International Application No
PCT/US 95/09691

A. CLASSIFICATION OF SUBJECT MATTER
IPC 6 G06F11/00

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
IPC 6 G06F

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practical, search terms used)

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category *	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	US,A,5 253 158 (SUZUKI ET AL.) 12 October 1993 see column 4, line 55 - column 5, line 8 ---	1-83
A	SYSTEMS & COMPUTERS IN JAPAN, vol.21, no.2, 1990, NEW YORK US pages 11 - 22 MIZUHITO OGAWA ET AL. 'Anomaly Detection of Functional Programs Based on Global Dataflow Analysis' see page 15, left column, line 1 - right column, line 10 --- -/--	1-83

[X] Further documents are listed in the continuation of box C.

[X] Patent family members are listed in annex.

* Special categories of cited documents :

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier document but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art.

"&" document member of the same patent family

1

Date of the actual completion of the international search	Date of mailing of the international search report
29 November 1995	18.12.95

Name and mailing address of the ISA	Authorized officer
European Patent Office, P.B. 5818 Patentlaan 2 NL - 2280 HV Rijswijk Tel. (+31-70) 340-2040, Tx. 31 651 epo nl, Fax: (+31-70) 340-3016	Corremans, G

Form PCT/ISA/210 (second sheet) (July 1992)

International Application No
PCT/US 95/09691

C.(Continuation) DOCUMENTS CONSIDERED TO BE RELEVANT

Category °	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	SOFTWARE PRACTICE & EXPERIENCE, vol.17, no.3, March 1987, CHICHESTER GB pages 227 - 239 FUN TING CHAN ET AL. 'AIDA - A Dynamic Data Flow Anomaly Detection System for Pascal Programs' see page 228, line 1 - line 21	1-83

1

INTERNATIONAL SEARCH REPORT

Information on patent family members

International Application No
PCT/US 95/09691

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
US-A-5253158	12-10-93	JP-A- 4218808	10-08-92
		KR-B- 9501058	08-02-95